



# 高级软件实作项目开发文档

题目：基于 SpringBoot 的食在生鲜超市系统

院系： 软件学院

年级： 19 级

专业： 软件工程

成员姓名及学号： 顾思睿 20192005273

黄海峰 20192005087

王哲孟 20192005343

何昕宇 20192005290

李牧林 20192005323

2022 年 7 月 25 日

# 目录

1.需求分析 .....	3
1.1 业务需求 .....	3
1.1.1 背景 .....	3
1.1.2 业务机遇 .....	3
1.1.3 业务目标 .....	3
1.1.4 成功指标 .....	3
1.1.5 愿景陈述 .....	3
1.1.6 业务风险 .....	4
1.1.7 业务假设与依赖 .....	4
1.2 范围与限制 .....	4
1.2.1 主要特性 .....	4
1.2.2 初始与后续发布的范围 .....	4
1.2.3 限制于排除项 .....	5
1.3 业务上下文 .....	5
1.3.1 干系人资料 .....	5
1.3.2 项目优先级 .....	5
1.4 页面设计 .....	6
1.4.1 主页 .....	6
1.4.2 个人中心 .....	6
1.4.3 购物车 .....	6
1.4.4 订单 .....	6
1.4.5 付款页 .....	6
1.4.6 数个具体类别页 .....	7
1.4.7 待收货 .....	7
1.4.8 注册登录 .....	7
1.5 业务规则与用例 .....	7
1.5.1 业务规则 .....	7
1.5.2 用例 .....	7
2.项目可行性分析 .....	9
2.1 技术可行性 .....	9
2.2 经济可行性 .....	9
2.3 法律可行性 .....	9
3.项目设计 .....	10
3.1 概要设计 .....	10
3.1.1 组织结构图 .....	10
3.1.2 功能结构图 .....	11
3.1.3 分析类图 .....	11
3.2 详细设计 .....	12
3.2.1 类图 .....	12
3.2.2 活动图 .....	13
4.项目实施 .....	18
4.1 前端 .....	18

4.2 后端.....	31
4.2.1 SpringCloud.....	31
4.2.2Mysql.....	41
4.2.3Redis.....	43
5.项目分工说明.....	48
5.1 组员分工 .....	48

# 1.需求分析

## 1.1 业务需求

### 1.1.1 背景

现阶段，国内的疫情形式依旧十分严峻，全国各地仍然存在各种规模的疫情爆发。对于那些正处于疫情防控区内的人，因为疫情而导致无法自由的出行，就意味着无法通过正常渠道来采购生活必需品，例如粮食、蔬果、肉类等，针对这种情况，我们计划开发一个网上生鲜商城，来为这些为了配合疫情防控的居家人员提供正规购买食材的渠道。

### 1.1.2 业务机遇

处于疫情管控区内的人们希望自己的生活质量能够得到保障。网上生鲜商城的出现正好为用户提供了这样一个平台。现在的防疫要求使管控区内的大部分人只能通过政府分配一定量的水和食物来生存，可对于有着特殊情况的家庭来说，是不合理的，该生鲜商城这可以一定程度上为居民提供足够的水和食品，借此保障居民的生活质量。与我们这个生鲜商城合作的商家们也可以通过这种方式来适当弥补疫情带来的经济损失。

### 1.1.3 业务目标

BO-1:在发布前的 6 个月，向各大商家宣传，进行合作。培养配送骑手。

BO-2:在发布后的 6 个月，累计用户数额，并继续扩大合作范围。

BO-3:在发布后的 12 个月，争取向其他疫区城市扩张发展。

### 1.1.4 成功指标

SM-1:在发布后的 6 个月，在本市的疫情管控区取得 50%的蔬菜、生鲜商家的合作。

SM-2:在发布后的 12 个月，能够满足本市疫情管控区内 80%居民的基本食材需要。

### 1.1.5 愿景陈述

对于希望想要弥补由于疫情带来的经济损失的商家来说，网上生鲜商城是一个基于互联网的 web 网页商城，它能够接受居民对于各种生鲜食材的购买，并进行支付处理，然后商家将食材通过骑手配送的方式配送到指定地点，使用网上生鲜商城可以帮助处于疫情管控区内的居民更方便的采购食材。

### 1.1.6 业务风险

- RI-1:在与各类符合生鲜商城要求的商家谈判时可能需要大笔资金来进行投入。(概率 =0.8; 影响=6)
- RI-2:若营业额较少的商家不同意合作，将降低人们使用生鲜商城的欲望。(概率=0.3; 影响=3)
- RI-3:如果在一段时间后使用生鲜商城的用户较少，会对该软件进行更多的需求开发和使商家经营重心的改变的投资回报率降低。(概率=0.3; 影响=7)

### 1.1.7 业务假设与依赖

- AS-1:生鲜商城为用户提供了恰当的 UI 和交互界面，使用户使用生鲜商城的频率上升。
- AS-2:在用户支付后，商家会在约定时间内将商品送达给客户。
- DE-1:如果合作的商家自身具有自己的系统可以处理客户的订单，生鲜商城就必须要与之能够形成双向通信。

## 1.2 范围与限制

### 1.2.1 主要特性

- FE-1:从生鲜商城中选择商品并进行加入购物车和支付
- FE-2:创建、查看、修改、删除以及存档各个商家在生鲜商城上显示的食材和展示的内容
- FE-3:查看商家上传的食材列表以及营养信息
- FE-4:授权顾客能够通过企业内网、智能手机、平板电脑以及外部互联网访问系统内部

### 1.2.2 初始与后续发布的范围

特性	发布 1	发布 2	发布 3
FE-1，购买商品	工作时间段配送到指定地点	支持各种移动支付，并且支持信用卡、花呗贷款支付	暂无
FE-2，商品列表	新增和编辑查看商品	修改、删除和存档商品	
FE-3，商品成分列表	未实现	完整实现	
FE-4，系统访问	内网和外部互联网	运行于 Windows 的	运行于 IOS 以及安卓

	访问	手机和电脑的应用	的手机和平板电脑的 APP
--	----	----------	---------------

### 1.2.3 限制于排除项

LI-1:生鲜商城的各个商家的商品列表上显示的必须是可以配送的食材,不可外送或者未达到规定时间不外送的食材应该以不可选的方式呈现在商品列表中。

LI-2:该网上生鲜商城在现阶段仅在项目开发的局域网内试行

## 1.3 业务上下文

### 1.3.1 干系人资料

干系人	主要价值	态度	主要兴趣	约束
商城运营管理层	决定商城未来的发展前景和方向	强烈承诺支持发布 2, 对于发布 3 的支持取决于之前的结果	对项目成本进行各种估算和统计	无明确约束
顾客	更便捷的生活方式、节约大量时间	热情高涨, 但可能会因为服务和价格问题减少使用	方便使用、节省时间	可以在任意设备上访问该平台
商家的服务人员	提高顾客的满意度, 扩大使用量	担心服务不周被顾客投诉	保住工作	对服务人员进行相关的系统培训
商家的管理层	更高的销售额, 通过曝光量来增加客户	利益至上, 一切均以收益为首要条件	赚钱	无明确约束

### 1.3.2 项目优先级

维度	约束	驱动	自由度
特性	所有排入发布 1 的		

	特性都必须完全可操作		
质量	用户验收测试的通过率必须超过 90%，各项安全测试必须全部通过		
成本			在无赞助方评审的情况下，可以接受不超过 20% 的预算超支
人员		项目开发团队包括 1 名前端人员,4 名后端人员	

## 1.4 页面设计

### 1.4.1 主页

实现展示外卖数个具体分类页面，比如美食的日料类别入口、西餐类别入口等等。还有部分热门美食推荐、搜索栏等组件。

### 1.4.2 个人中心

实现用户个人中心页面，用户可以在此页面更改个人信息如称呼手机号等等，还可以在此页面查看一些辅助功能如待收货列表。

### 1.4.3 购物车

用户选择的餐品放入购物车中暂存，最后用于结算。

### 1.4.4 订单

展示用户之前已经选购结算的餐品的具体信息。

### 1.4.5 付款页

此页面完成购物车餐品的结算业务。

## 1.4.6 数个具体类别页

展示具体的某个餐饮类别（如西餐、日料、韩料）的商店列表，供用户选择。

## 1.4.7 待收货

展示用户已付款但未到货的餐品情况。

## 1.4.8 注册登录

完成一个用户的登录注册业务。

## 1.5 业务规则与用例

### 1.5.1 业务规则

- BR-1：用户需登录账号才能使用商城功能。
- BR-2：用户登录时，如果密码错误，则需重新输入新的验证码。
- BR-3：只有具有管理员权限的账号才能进入后台管理。
- BR-4：商品库存为零时，用户无法购买该商品。
- BR-5：订单超过 2 小时未支付则取消订单。

### 1.5.2 用例

#### 主要操作者和用例

主要操作者	用例
用户	1. 注册 2. 登陆 3. 查看商品 4. 搜索商品 5. 加入购物车 6. 删除购物车 7. 创建订单 8. 删除订单 9. 支付
商家	1. 注册 2. 登录





## 2.项目可行性分析

### 2.1 技术可行性

- a、风险分析：本实作有固定人员 5 人，不会造成人员流失。本实作成品开发与管理使用 springboot、Maven、redis、mysql、javascript、springCloud、Nginx 技术，此为成熟的技术，所以并不会会有技术性风险。
- b、资源分析：本项目使用 eclipse 作为 IDE 进行开发，搭建一个服务性网站，现有的软件和硬件都能支撑项目开发。开发人员已经接受了前后端开发的培训，熟悉开发技术。
- c、技术分析：
  - 1)易用性：基于 springboot 框架开发的前后端服务性网站，能保证服务器正常运作下 99%能成功进入系统。
  - 2)性能：基于 express 框架开发的网站能承担大约 30 条并发访问。系统能够存储大约 1000 条用户信息。
  - 3)防护性：采用 post 请求模式保证用户的数据在网站不被公开。用户注册登录信息、反馈信息、浏览信息存入本地数据库后保证不被泄露。
  - 4)健壮性：网站设计时，开发人员会考虑各种报错情况并及时给予错误反馈，所以网站遇到非法输入时，能及时做出正确反映，保证系统正常运行而不是崩溃。

### 2.2 经济可行性

- a、成本：项目开发中所使用的软件均为免费版，硬件为学生本人计算机，同时使用华为云服务器作为 mysql 与 redis 数据库服务器。开发人员为学生故无开发人力成本。开发人员已接受培训，无成本。学生组队进行自我管理，没有管理成本。
- b、效益：此项目为学院实作项目作业，不产生经济效益。
- c、投资回收期：因为没有经济效益，故没有投资回收期。
- d、纯收入：不会产生收入。

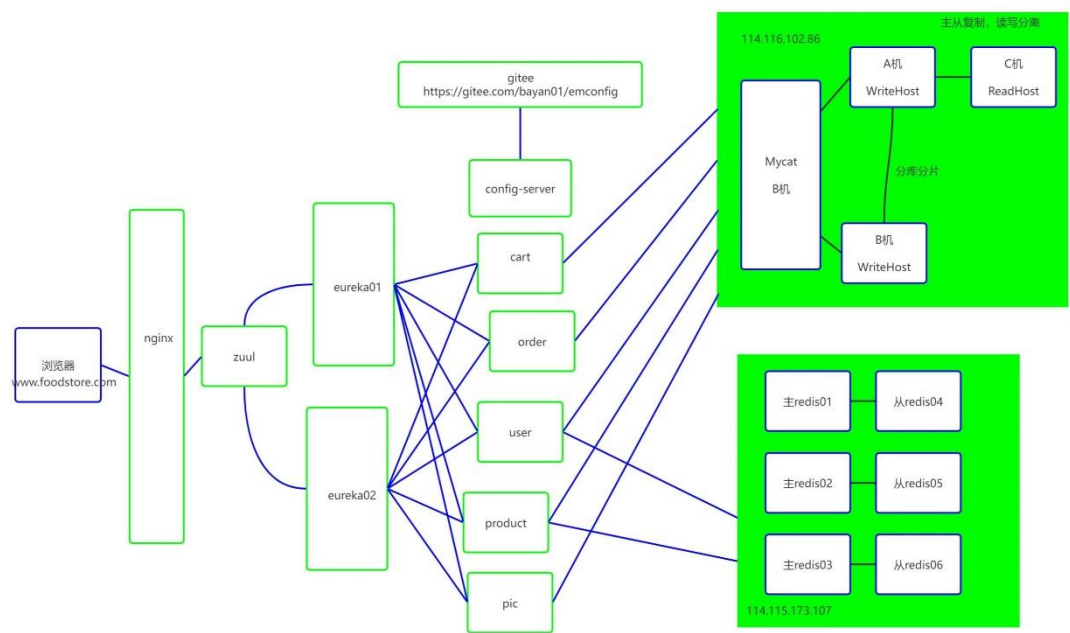
### 2.3 法律可行性

- a、此实作所使用的数据为网络公开数据，无侵权行为。
- b、此实作的内容积极、向上、健康、非盈利，在法律的运行范围中。

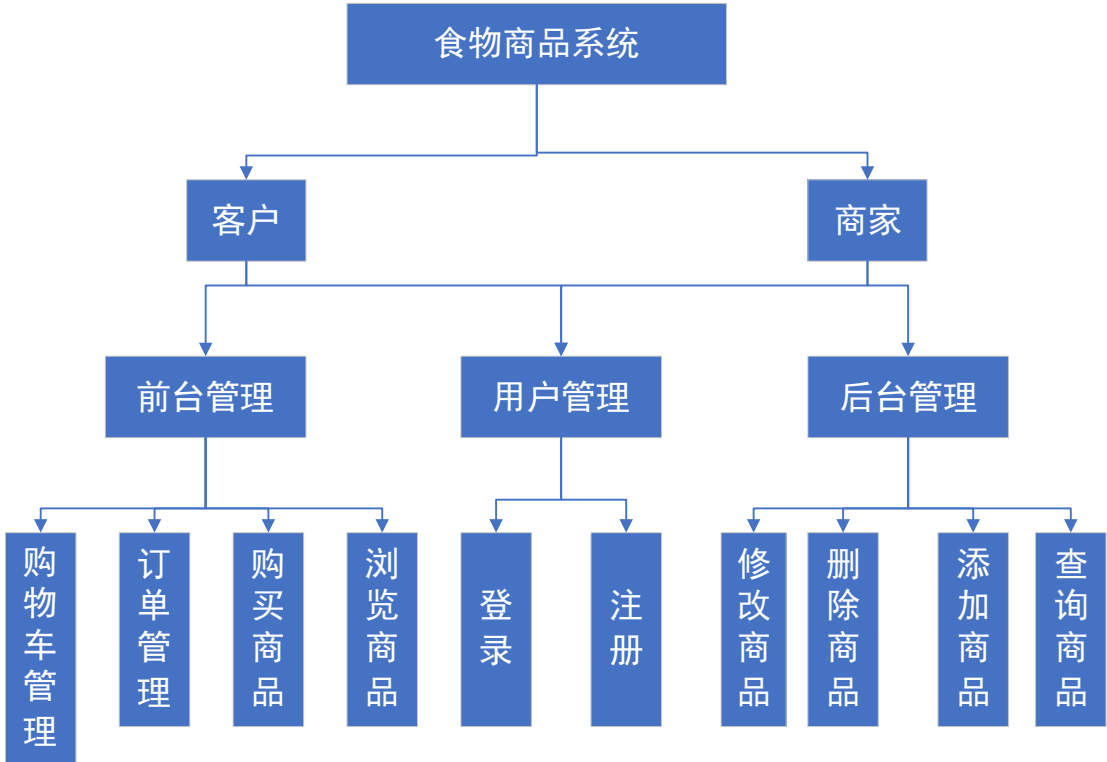
# 3.项目设计

## 3.1 概要设计

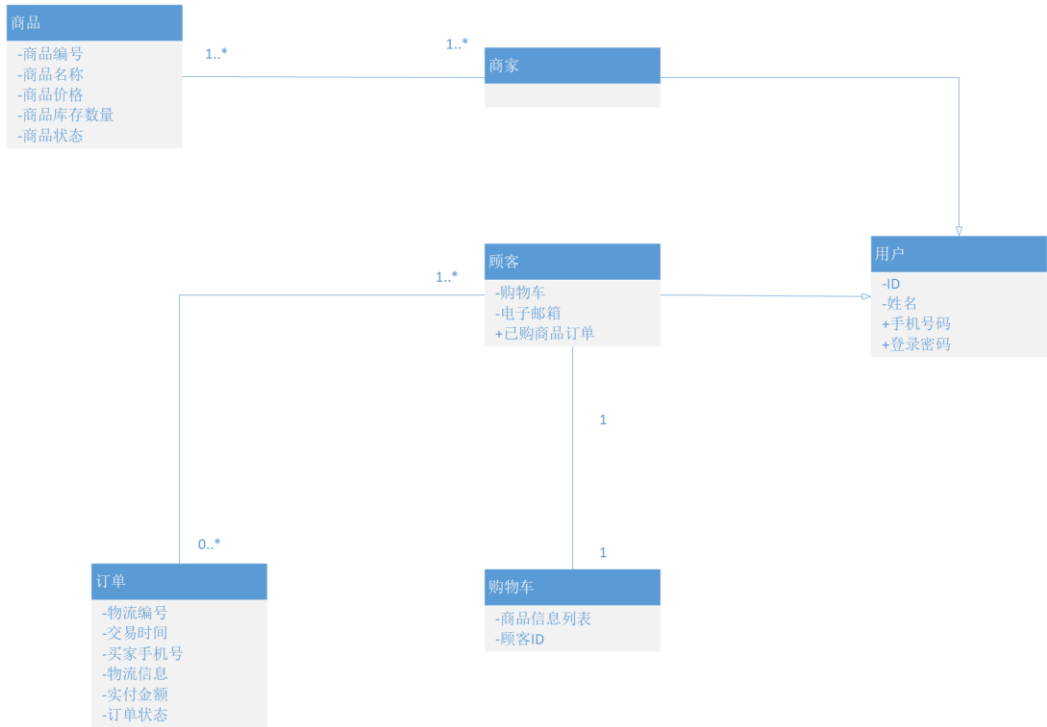
### 3.1.1 组织结构图



3.1.2 功能结构图

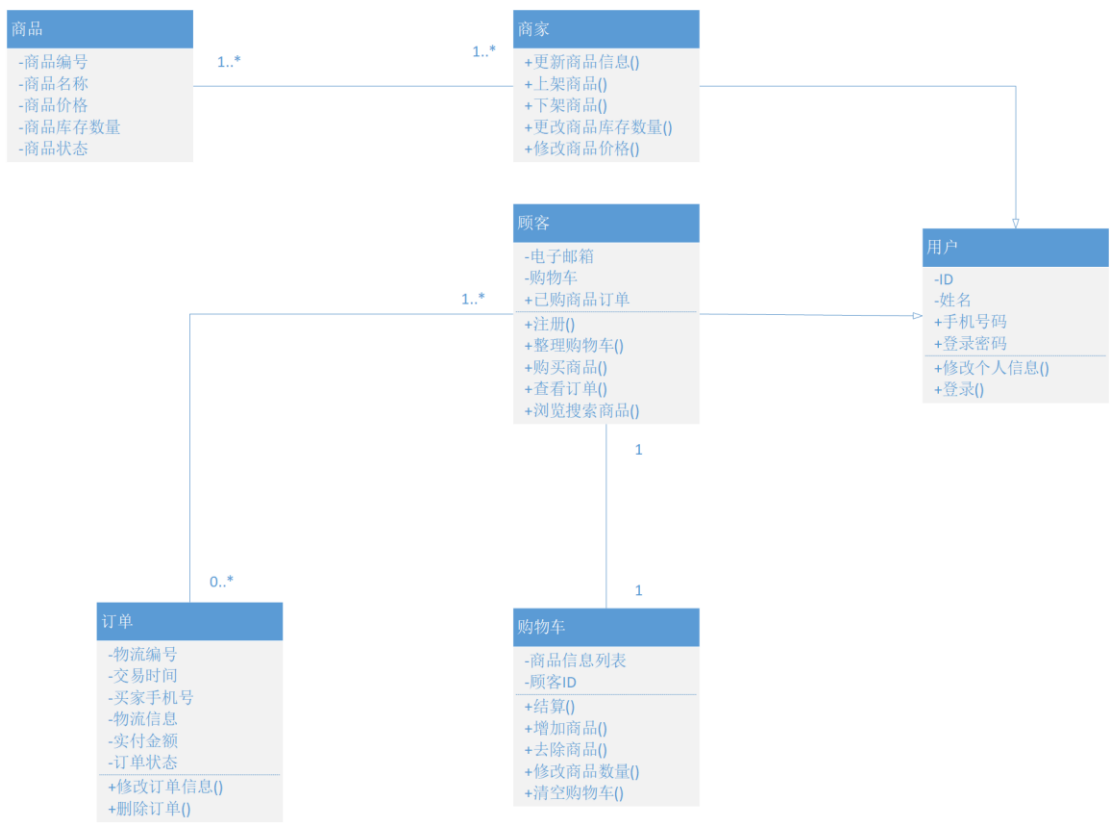


3.1.3 分析类图



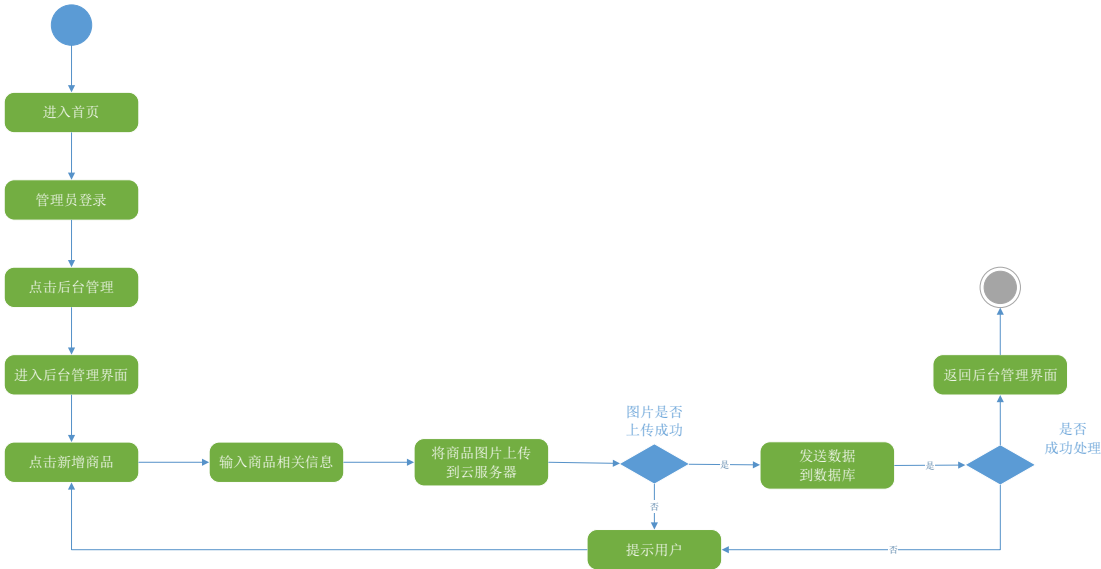
## 3.2 详细设计

### 3.2.1 类图

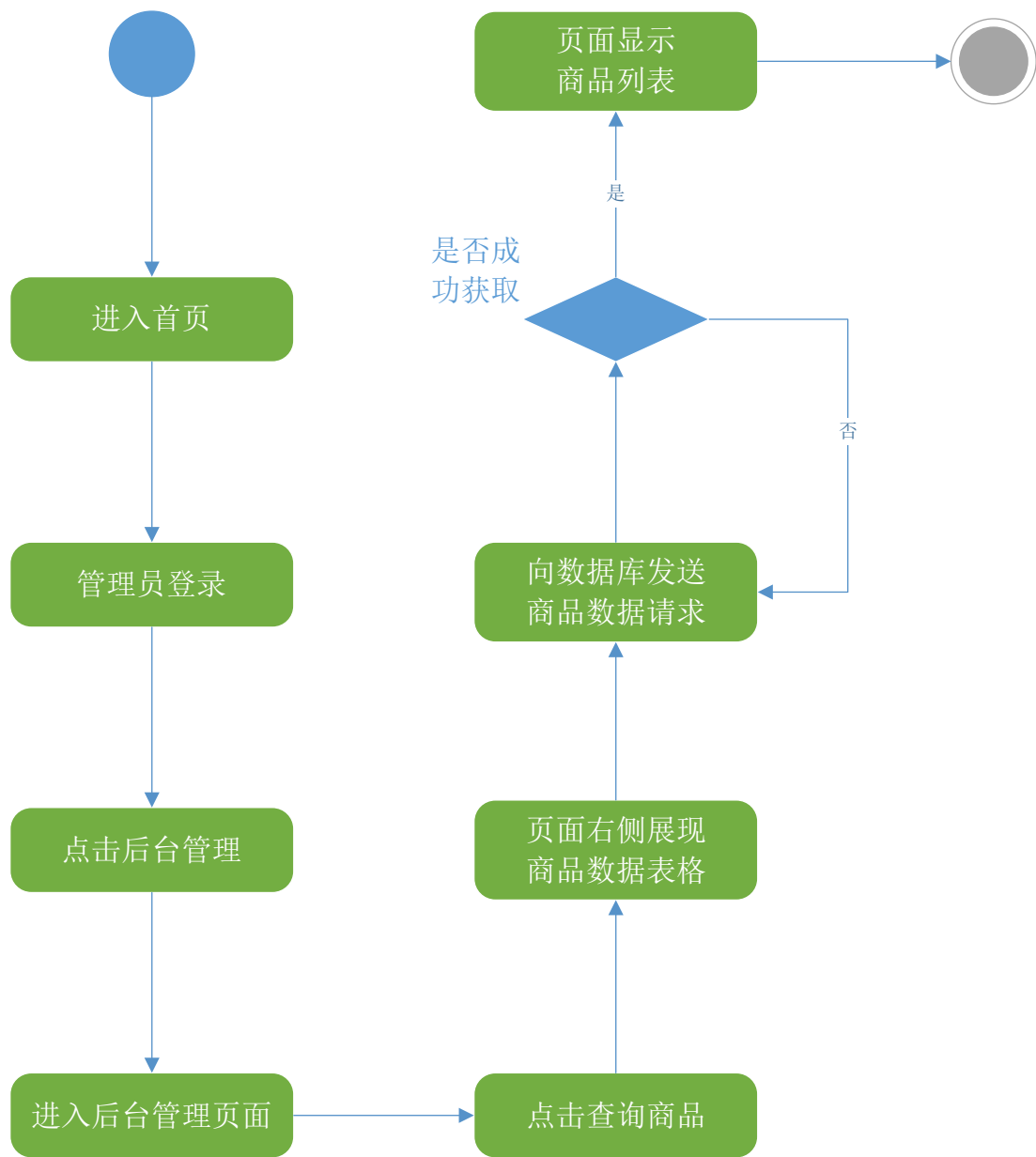


### 3.2.2 活动图

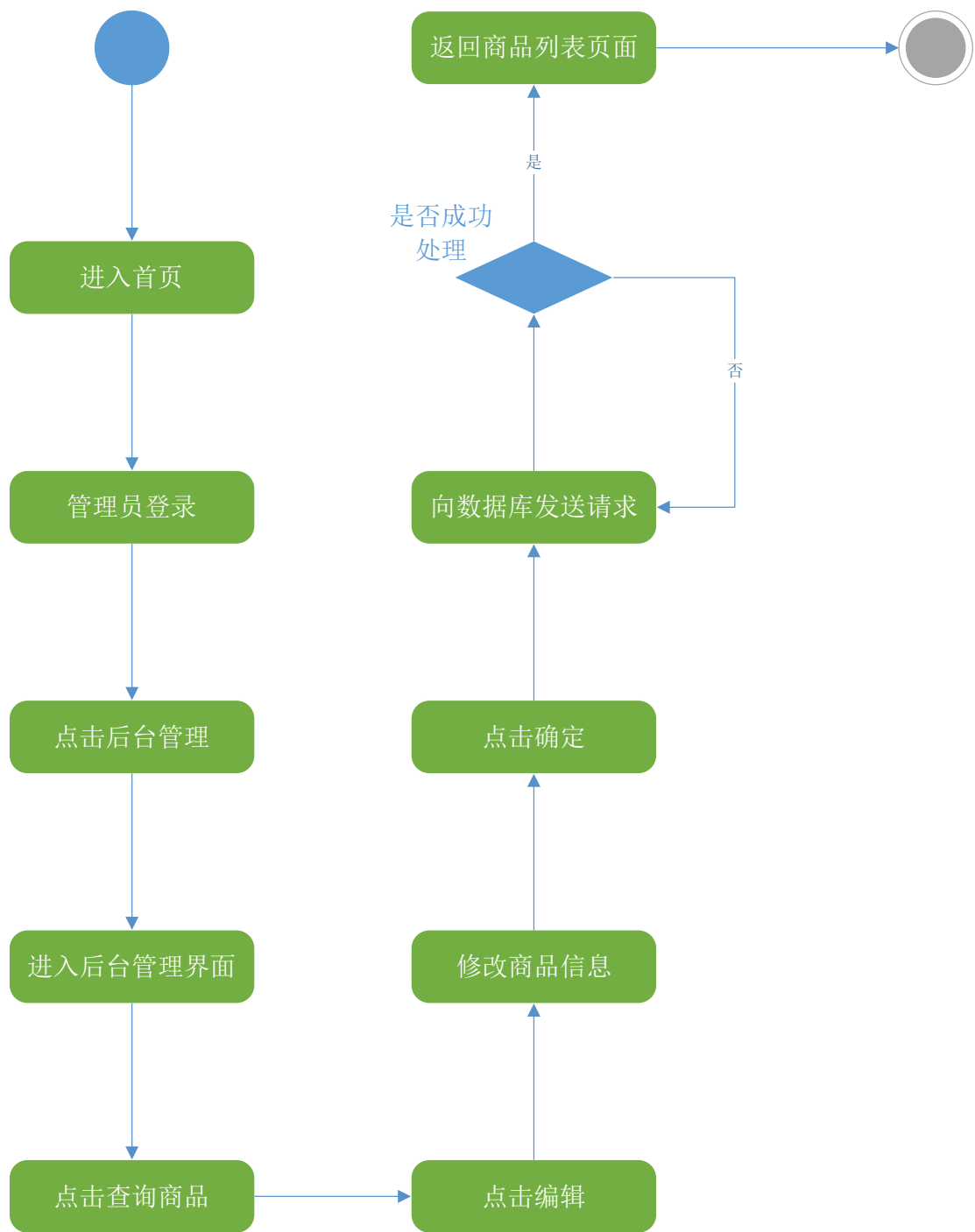
#### 3.2.2.1 后台管理活动图



商品新增活动图

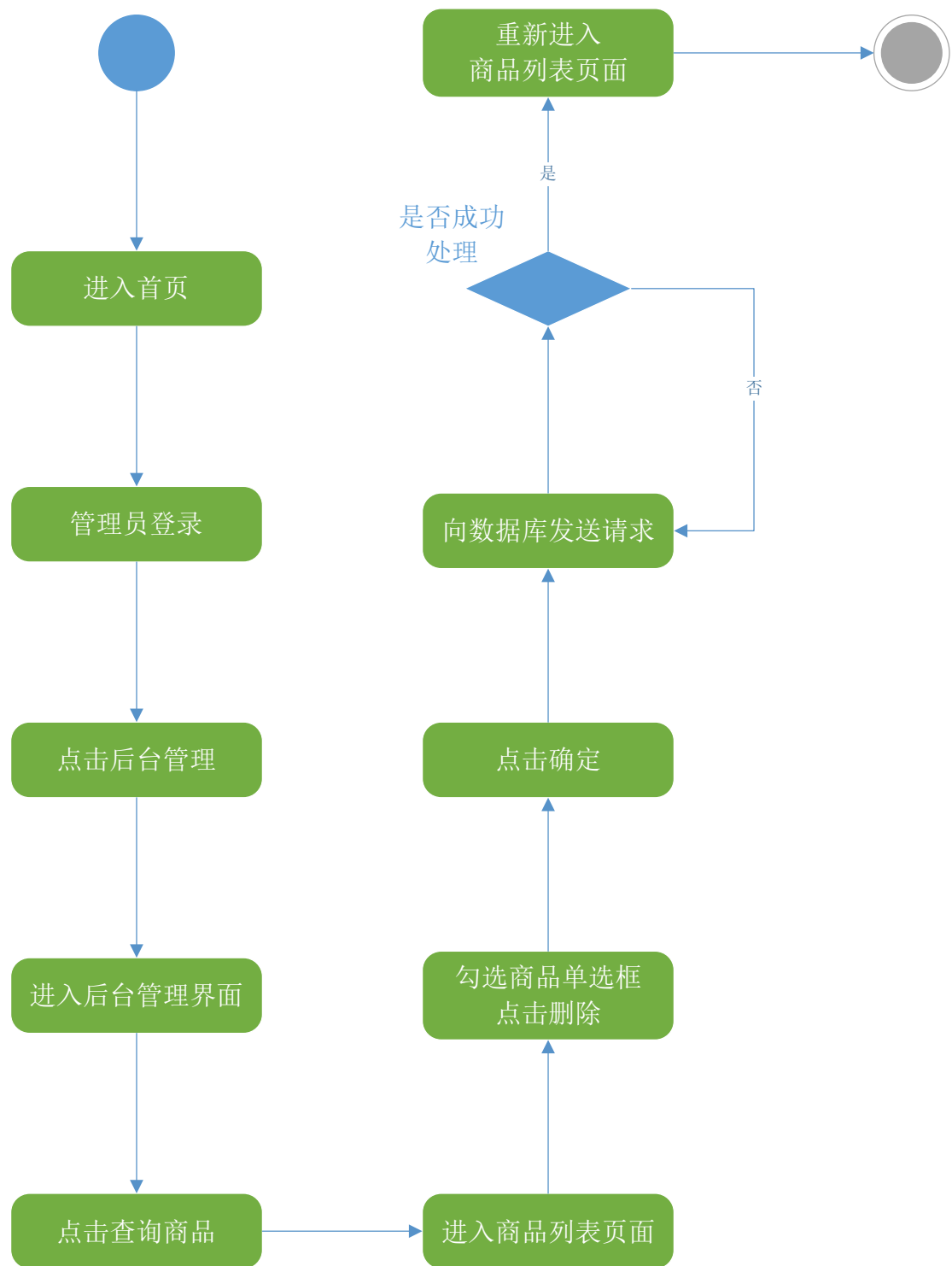


商品查询活动图



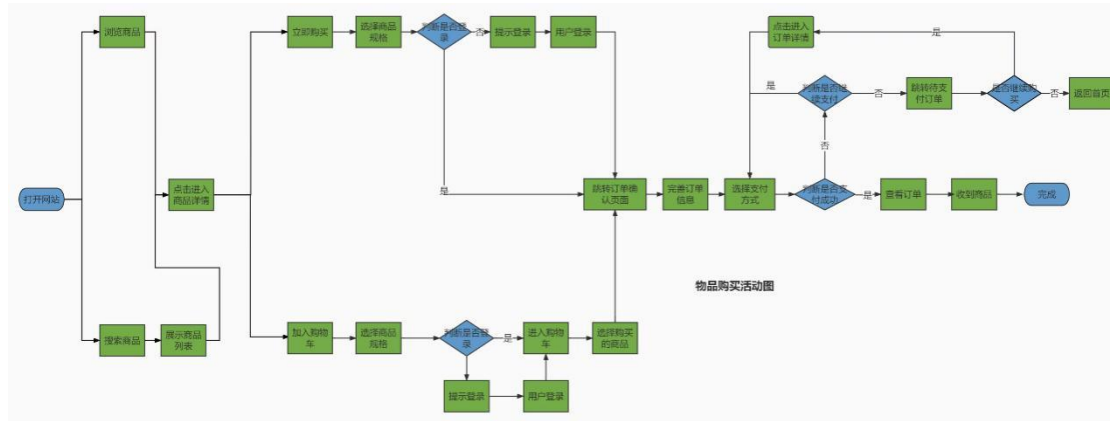
商品信息编辑活动图





商品删除活动图

### 3.2.2.2 物品购买活动图



## 4.项目实现

### 4.1 前端

该 FoodStore 生鲜商城系统总共分为以下几个模块：

商城 index 主页、添加商品的 manage 管理页面、商品页、登录注册页、购物车和订单详情页。

这些模块 html 页面和各个模块所需要的 css 文件和 jsp 文件，共同构成了我们 FoodStore 生鲜商城的前端页面。接下来就由我来简单分析一下各个模块的相关代码。

商城 index 主页：

```
<div id="line3">
  <div id="content">
    <ul>
      <li><a href="./index.html">首页</a></li>
      <li><a href="./product-list.html?page=1&rows=30">全部商品</a></li>
      <li><a href="./product-list-other.html?page=1&rows=20&category=新鲜水果">新鲜水果</a></li>
      <li><a href="./product-list-other.html?page=1&rows=20&category=海鲜水产">海鲜水产</a></li>
      <li><a href="./product-list-other.html?page=1&rows=20&category=精选肉类">精选肉类</a></li>
      <li><a href="./product-list-other.html?page=1&rows=20&category=冷冻饮食">冷冻饮食</a></li>
      <li><a href="./product-list-other.html?page=1&rows=20&category=蔬菜蛋品">蔬菜蛋品</a></li>
    </ul>
  </div>
</div>
```

目的是展示我们商城的大致情况，主要内容是 content 的目录内容，可以点击并链接到对应的商品分类页中，实现对商品的筛选，方便用户在使用过程中进行商品的购物。



添加商品的 manage 管理页面：

```

<title>FoodStore商城后台管理系统</title>
<style type="text/css">
    .content {
        padding: 10px 10px 10px 10px;
    }
</style>
</head>

<body class="easyui-layout">
    <div data-options="region:'west',title:'菜单',split:true" style="width:180px;">
        <ul id="menu" class="easyui-tree" style="margin-top: 10px;margin-left: 5px;">
            <li>
                <span>商品管理</span>
                <ul>
                    <li data-options="attributes:{'url':'./item-add.html'}">新增商品</li>
                    <li data-options="attributes:{'url':'./item-list.html'}">查询商品</li>
                </ul>
            </li>
        </ul>
    </div>
    <div data-options="region:'center',title:''">
        <div id="tabs" class="easyui-tabs">
            <div title="首页" style="padding:20px;">
            </div>
        </div>
    </div>

    <script type="text/javascript">
        $(function () {
            $('#menu').tree({
                onClick: function (node) {
                    if ($('#menu').tree("isLeaf", node.target)) {
                        var tabs = $('#tabs');
                        var tab = tabs.tabs("getTab", node.text);
                        if (tab) {
                            tabs.tabs("select", node.text);
                        } else {
                            tabs.tabs('add', {
                                title: node.text,
                                href: node.attributes.url,
                                closable: true,
                                bodyCls: "content"
                            });
                        }
                    }
                }
            });
        });
    </script>

```

该 manage 管理页面主要是为了能够对商品进行添加和修改，需要手动输入 [www.foodstore.com/manage.html](http://www.foodstore.com/manage.html) 进行访问。在该页面中能够跳转到对应的 item-add 页和 item-list 页，实现对商品的修改、添加和删除。

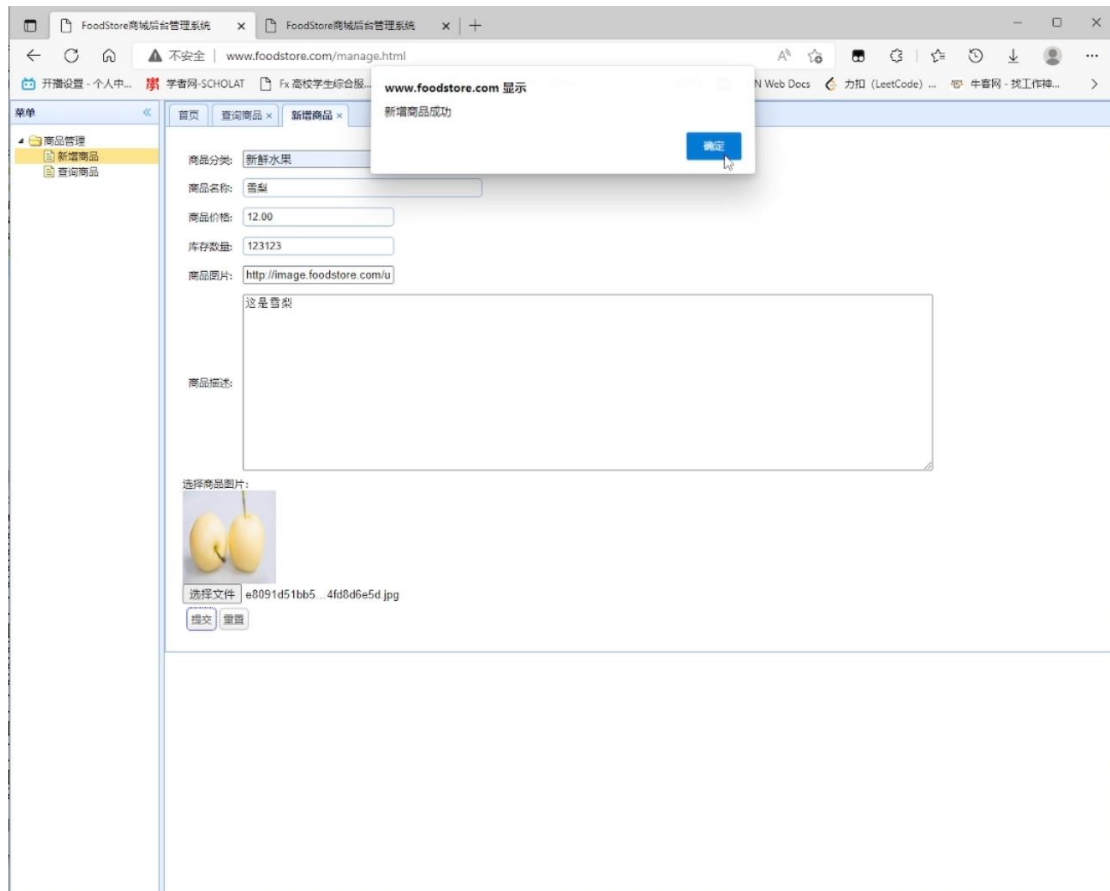
在 item-add 页面中，主要是实现商品添加的功能

```

function submitForm() {
    alert($("#itemAddForm").serialize());
    $.post("/products/save", $("#itemAddForm").serialize(), function (data) {
        if (data.status == 200) {
            alert("新增商品成功");
        }
    });
}

```

新增商品的表单数据作为参数, jquery 将其序列化成了 key=value&key=value 的字符串,在请求体中提交,接收返回的数据,判断执行状态结果弹框提示。



在 item-edit 页面中，主要是实现修改的功能

```
function submitForm(){
    if(!$('#itemEditForm').form('validate')){
        $.messager.alert('提示','表单还未填写完成!');
        return ;
    }
    alert($("#itemEditForm").serialize())
    $.post("/products/update",$("#itemEditForm").serialize(), function(data){
        if(data.status == 200){
            $.messager.alert('提示','修改商品成功!','info',function(){
                $("#itemEditWindow").window('close');
                $("#itemList").datagrid("reload");
            });
        }else{
            alert(data.msg);
        }
    });
}
```

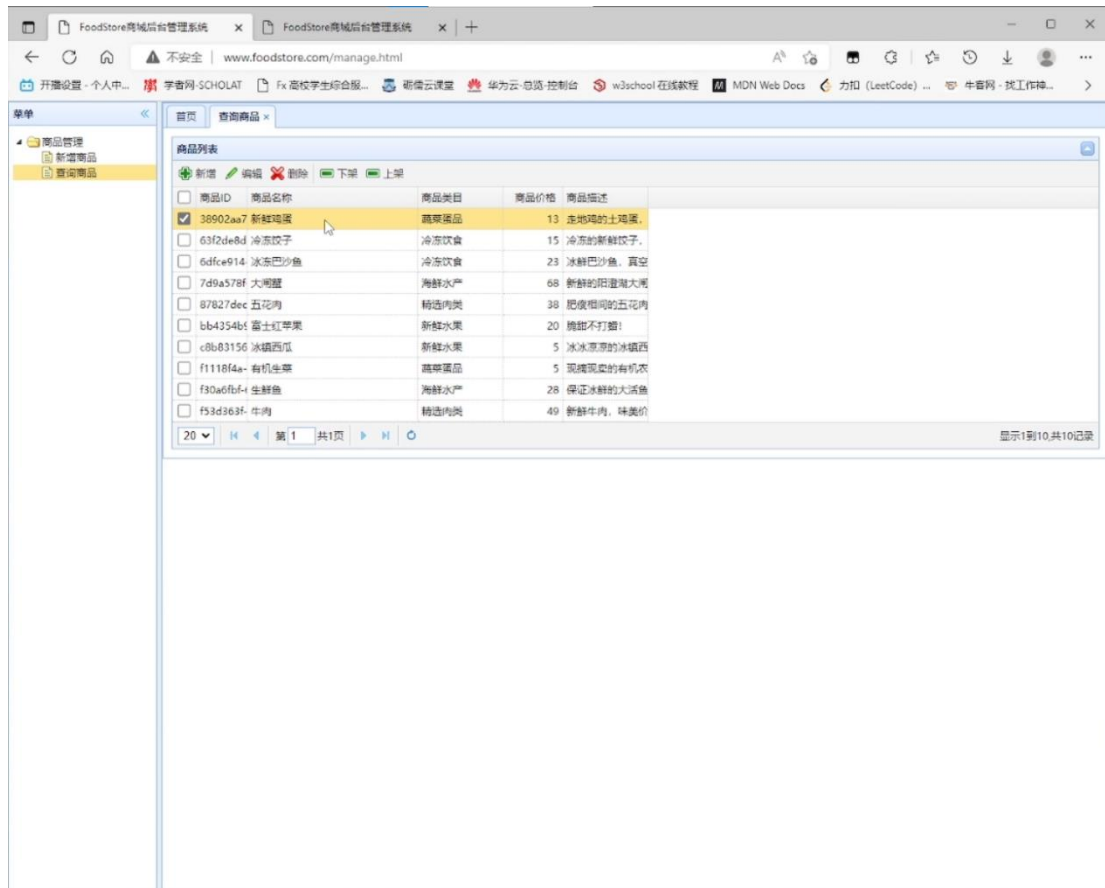
这里是将修改商品的表单数据作为参数，后面的逻辑和上述商品添加类似，都是通过 js 将参数提交给后端接口，再接收后端返回的数据进行判断。

在 item-list 页面中，主要是实现查询商品的功能，可以看到所有已经在数据库中的商品信息

息，相关代码如下：

```
var toolbar = [{
    text: '新增',
    iconCls: 'icon-add',
    handler: function () {
        $(".tree-title:contains('新增商品')").parent().click();
    }
}, {
    text: '编辑',
    iconCls: 'icon-edit',
    handler: function () {
        var ids = getSelectionsIds();
        if (ids.length == 0) {
            $.messager.alert('提示', '必须选择一个商品才能编辑!');
            return;
        }
        if (ids.indexOf(',') > 0) {
            $.messager.alert('提示', '只能选择一个商品!');
            return;
        }

        $("#itemEditWindow").window({
            onLoad: function () {
                //回显数据
                var data = $("#itemList").datagrid("getSelections")[0];
                $("#itemEditForm").form("load", data);
                KindEditorUtil.init({
                    "pics": data.productImage,
                    "cid": data.cid,
                    fun: function (node) {
                        KindEditorUtil.changeItemParam(node,
                            "itemEditForm");
                    }
                });
            }
        }).window("open");
    }
}, {
    text: '删除',
    iconCls: 'icon-cancel',
```



商品页：

商品页分为商品信息页和商品列表页，其中由于后端函数的不同，商品列表页分为 2 类，一类是只有 page 和 rows 两个参数的全部商品列表，另一个是有 page、rows 和 category 三个参数的分类商品筛选列表。

商品信息页：点击查询所选中的单个商品。

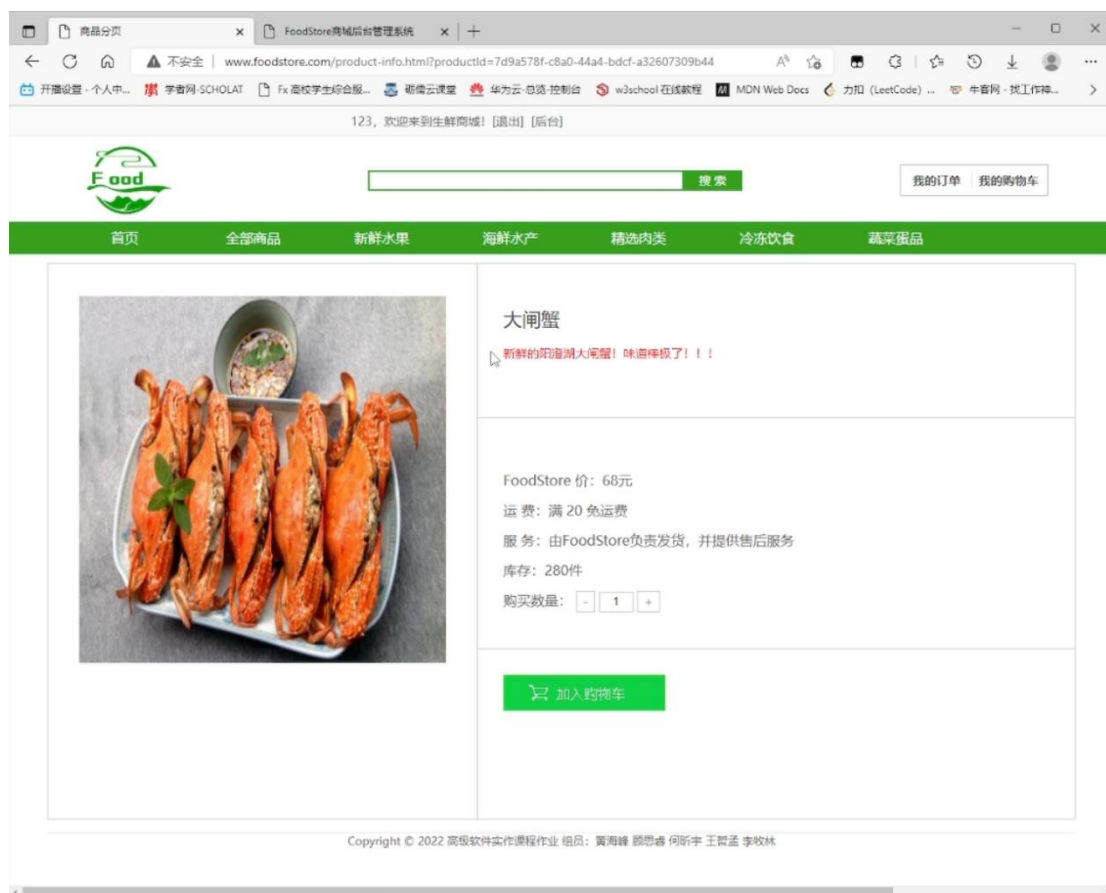
```

window.onload = function () {

    var localUrl = decodeURI(window.location.href);
    //alert(localUrl);
    productId = localUrl.split("?")[1].split("=")[1];
    $.ajax({
        url: "http://www.foodstore.com/products/item/" + productId,
        dataType: "json",
        type: "POST",
        success: function (data) {
            //图片
            var imgUrl = data.productImgurl;
            productId = data.productId;
            document.getElementById("prod_img").src = imgUrl;
            document.getElementById("prod_name").innerText = data.productName;
            document.getElementById("prod_desc").innerText = data.productDescription;
            document.getElementById("prod_pric").innerText = data.productPrice;
            document.getElementById("prod_num").innerText = data.productNum;
        }
    });
    var _ticket = $.cookie("EM_TICKET");
    //alert(_ticket);
    if (!_ticket) {
        return;
    }
}

```

主要逻辑是在页面加载后直接调用 ajax 请求,将返回的对象数据,解析,调用 js 代码=拼接到请求地址中后,方便后端接收到 productId 并返回所查询的 product 对象 json 字符串。

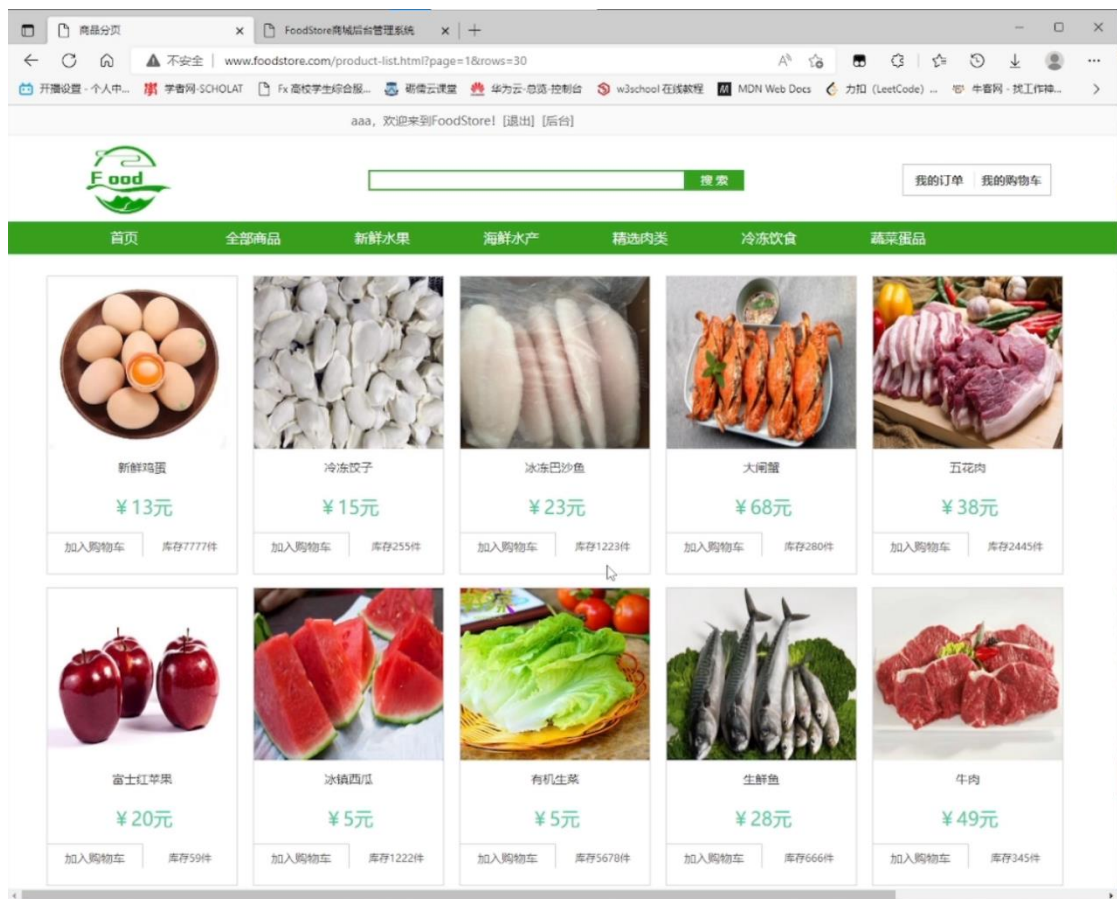




商品列表页：将对应的商品展现在网页上。

```
function queryPages(param) {
    var ticket = $.cookie("EM_TICKET");
    var userId = "";
    $.get("http://www.foodstore.com/user/query/" + ticket, function (data) {
        if (data.status == 200) {
            var _data = JSON.parse(data.data); //jackson
            userId = _data.userId;
        }
    })
    sleep(200);
    //alert("等待当前页面加载userId");
    $.ajax({
        url: "http://www.foodstore.com/products/pageManage?" + param,
        dataType: "json",
        type: "POST",
        success: function (data) {
            if (data.rows.length > 0) {
                $.each(data.rows, function (index, product) {
                    var product_html = "product-info.html?productId=" + product.productId;
                    var productId = product.productId;
                    $("#prod_content").append("<div id='prod_div'><a href=" + product_html +
                        "></img> </a><div id='prod_name_div'><a href=" +
                        product_html + ">" + product.productName +
                        "</a></div><div id='prod_price_div'>¥" + product.productPrice +
                        "元</div><div id='gotocart_div'><a href='javascript:void(0)' onclick='addCartOne(\"" +
                        productId + "\",\"" + userId +
                        "\")'>加入购物车</a></div><div id='say_div'>库存" + product
                        .productNum + "件</div></div></div>");
                })
            }
        },
        error: function () {
            alert("请求失败");
        }
    });
}
```

主要逻辑是通过 queryPages 函数接收数据库中插入的商品数据，如商品的 id、img 的存储 url、name、price 等信息。并通过 append 函数将其动态的生成在 prodlist 页面，提供给用户进行交互。



登录注册页：让用户能够注册并登录我们的 foodstore 生鲜商城  
Login\_ajax.js

```

$(function(){
    //给form表单添加submit事件
    $("form").submit(function(){
        return login();
    });
});
function login(){
    //获取页面数据
    var userName=$("form input[name=username]").val();
    var userPassword=$("form input[name=password]").val();
    if(userName==""){
        $("form table tr:eq(0) td span").html("用户名不能为空");
        return false;
    }
    if(userPassword==""){
        $("form table tr:eq(1) td span").html("密码不能为空");
        return false;
    }

    $.ajax({
        url:"http://www.foodstore.com/user/login",
        type:"get",
        data:{"userName":userName,"userPassword":userPassword},
        dataType:"json",
        success:function(result){
            //result是服务端返回的数据
            if(result.status==200){
                //window.location.href="index.html";
                window.location.href="index.html";
            }else{
                alert("登录失败");
            }
        },
        error:function(){
            alert("请求失败!");
        }
    });

    return false;
}

```

从表单中读取用户名和密码，传到服务端进行验证。通过接收服务端返回的数据来判断是否登录成功，若成功则跳转到 index 首页，不成功则弹出登录失败。



regist\_ajax.js

```
function register(){
    var userName=$("#form input[name=username]").val();
    var userPassword=$("#form input[name=password]").val();
    var userPassword2=$("#form input[name=password2]").val();
    var userNickName=$("#form input[name=nickname]").val();
    var userEmail=$("#form input[name=email]").val();
    //var vcode=$("#form input[name=valistr]").val();
    var flag=formObj.checkForm();
    if(flag){
        $.ajax({
            url:"/user/save",
            type:"post",
            data:{
                "userName":userName,
                "userPassword":userPassword,
                "userNickname":userNickName,
                "userEmail":userEmail,
            },
            dataType:"json",
            success:function(result){
                if(result.status==200){
                    alert("注册成功,转向登录页面")
                    window.location.href="/login.html";
                }else{
                    alert(result.msg);
                }
            },
            error:function(){
                alert("请求失败! ");
            }
        });
    }
    return false;
}
```

从表单中读取各项输入的数据，例如用户名、用户密码等，然后利用 checkForm 函数对输入的数据进行判断，若数据格式不符合要求则显示出相对于的警告。如果各项数据格式没有出现问题，就通过 url 提交到服务端，再通过服务端 result 的状态判断是否注册成功。



欢迎注册FoodStore

用户名:  用户名不能为空

密码:

确认密码:

昵称:

邮箱:

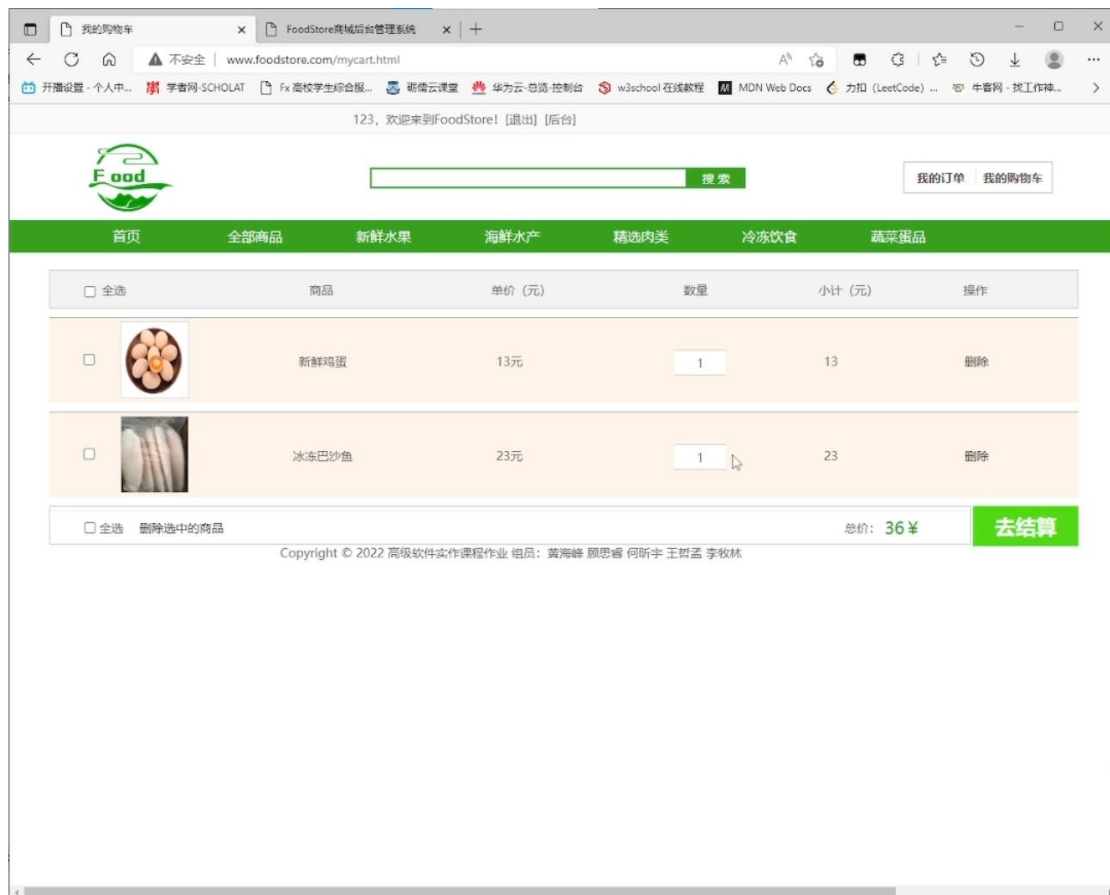
购物车和订单详情页：进行将商品加入购物车并支付完成后生成订单的功能编写

```
function query(userId) {
    $.ajax({
        url: "http://www.foodstore.com/cart/query?userId=" + userId,
        dataType: "json",
        type: "GET",
        success: function (data) {
            if (data.length > 0) {
                var money = 0;
                $.each(data, function (index, cart) {
                    var productId = cart.productId;
                    var productName = cart.productName;
                    var productPrice = cart.productPrice;
                    var productImage = cart.productImage;
                    var num = cart.num;
                    var prodHtml = "product-info.html?productId=" + productId;
                    var inputId = "hid_" + productId;
                    var itemMoney = productPrice * num;
                    money = money + itemMoney;

                    $("#carts").append(
                        "<div id='prods'><input name='prodC' type='checkbox' value=''/><a href='" +
                        prodHtml + "'><img src='" + productImage +
                        "' width='80' height='90'/></a><a href='" + prodHtml +
                        "'><span id='prods_name'>" + productName +
                        "</span></a><span id='prods_price'>" + productPrice +
                        "元</span><span><input type='hidden' id='" + inputId +
                        "' value='" + num + "'><input class='buyNumInp1' id='" +
                        productId + "' type='text' value='" + num +
                        "' onblur='inputNum(\"" + productId + "\",\"" + userId +
                        "\")'></span><span id='prods_money'>" + itemMoney +
                        "</span><span id='prods_del'><a href='javascript:void(0)' class='delNum' onclick='del" +
                        productId + "\",\"" + userId + "\">删除</a></span></div>";
                    });
                    document.getElementById("span_2").innerText = money + "¥";
                    //alert(money);
                } else {
                }
            }
        }
    });
}
```

此处是购物车的主要逻辑，意思是根据指定的 url 地址利用 GET 方式请求后端数据，请求成功后通过 success 的 data 形参数来接收数据，再将接收到的数据分别赋值给对应的属性值，

最后通过 append 函数来动态生成购物车的页面




订单详情页的逻辑和购物车的主要逻辑类似，只不过后端返回的是和订单相关的数据，然后在生成订单详情页面的时候需要书写不同的标签等

我的订单

FoodStore商城后台管理系统

www.foodstore.com/myorder.html

123, 欢迎来到生鲜商城! [退出] [后台]



搜索

我的订单

我的购物车

首页

全部商品

新鲜水果

海鲜水产



精选肉类

冷冻饮食

蔬菜蛋品

订单信息

订单编号:68965335-8aaa-4f3b-a598-d65672816083  
下单时间:1658555422000  
订单金额:36  
支付状态:已支付  
收货地址:123  
支付方式:在线支付

商品图片	商品名称	商品单价	购买数量	小计
	新鲜鸡蛋	13元	1件	13元
	冰冻巴沙鱼	23元	1件	23元

36元

Copyright © 2022 高级软件实训课程作业 组员: 黄海峰 顾思睿 何昕宇 王哲孟 李牧林







## 4.2 后端

### 4.2.1 SpringCloud

#### 1. SpringCloud 框架中依赖资源的管理

##### a) foodstore-parent

- ▼  foodstore-parent [FoodStore master]
  - >  JRE System Library [J2SE-1.5]
  - >  Maven Dependencies
  -  pom.xml

i.

- ii. foodstore-parent 作为父级工程, 合理管理资源。其继承了 springboot-parent, 导入 springcloud 资源, 并引入了项目中需要用到的公用依赖, 包括 eureka 和 ribbon 等依赖。

##### b) foodstore-common-repository






























- ▼  foodstore-common-repository [FoodStore master]
  - ▼  src/main/java
    - ▼  cn.edu.scnu.repository.config
      - >  DataSourceInitConfig.java
    - >  JRE System Library [JavaSE-1.8]
    - >  Maven Dependencies
    - >  src
    -  target
    -  pom.xml

i.

- ii. 该工程继承了 foodstore-parent, 引入了 jdbc, mysql, mybatis, druid 数据源连接池依赖。创建了自定义初始化属性的配置类。


























##### c) foodsotre-common-resources



- ▼  foodstore-common-resources [FoodStore master]
  - ▼  src/main/java
    - ▼  com.easymall.common.pojo
      - >  Cart.java
      - >  Order.java
      - >  OrderItem.java
      - >  Product.java
      - >  Seckill.java
      - >  Success.java
      - >  User.java
    - ▼  com.easymall.common.utils
      - >  CookieUtils.java
      - >  ListTransUtils.java
      - >  MapperUtil.java
      - >  MD5Util.java
      - >  RabbitmqUtils.java
      - >  UploadUtil.java
      - >  UUIDUtil.java
    - ▼  com.easymall.common.vo
      - >  EasyUIResult.java
      - >  HttpResult.java
      - >  Page.java
      - >  PicUploadResult.java
      - >  SysResult.java
    - >  JRE System Library [JavaSE-1.8]
    - >  Maven Dependencies
    - >  src
    - >  target
    - >  pom.xml

- i.
- ii. 该工程继承了 foodstore-parent, 维护项目运行所需工具类、视图类与 pojo 类。












## 2. 高可用双注册中心的搭建

- ▼  > easymall [FoodStore master]
  - >  config-client [FoodStore master]
  - >  config-server [FoodStore master]
  - ▼  > eureka-server [FoodStore master]
    - ▼  src/main/java
      - ▼  cn.edu.scnu
        - >  StarterEurekaServer01.java
    - ▼  src/main/resources
      -  application.properties
    - >  JRE System Library [JavaSE-1.6]
    - >  Maven Dependencies
    - >  src
    - >  > target
    -  pom.xml
  - ▼  > eureka-server2 [FoodStore master]
    - ▼  src/main/java
      - ▼  cn.edu.scnu
        - >  StarterEurekaServer02.java
    - ▼  src/main/resources
      -  application.properties
    - >  JRE System Library [JavaSE-1.6]
    - >  Maven Dependencies
    - >  src
    - >  > target
    -  pom.xml

a)





















- b) Eureka-server 与 eureka-server2 是相互注册的注册中心，只有端口号不同。以 eureka-server 为例，其继承了 spring-boot-starter-parent，引入了 spring-cloud-dependencies 管理的依赖，以及 spring-cloud-starter-eureka-server 依赖，用于启动注册中心服务器。编写了启动类，并在 application.properties 中进行了属性配置。

### 3. 高可用 zuul 网关集群的实现

- ▼  > foodstore-micro-zuul [FoodStore master]
  - ▼  src/main/java
    - ▼  cn.edu.scnu
      - >  StarterZuul.java
  - ▼  src/main/resources
    -  application.properties
  - >  JRE System Library [JavaSE-1.6]
  - >  Maven Dependencies
  - >  src
  - >  > target
  - a)  pom.xml

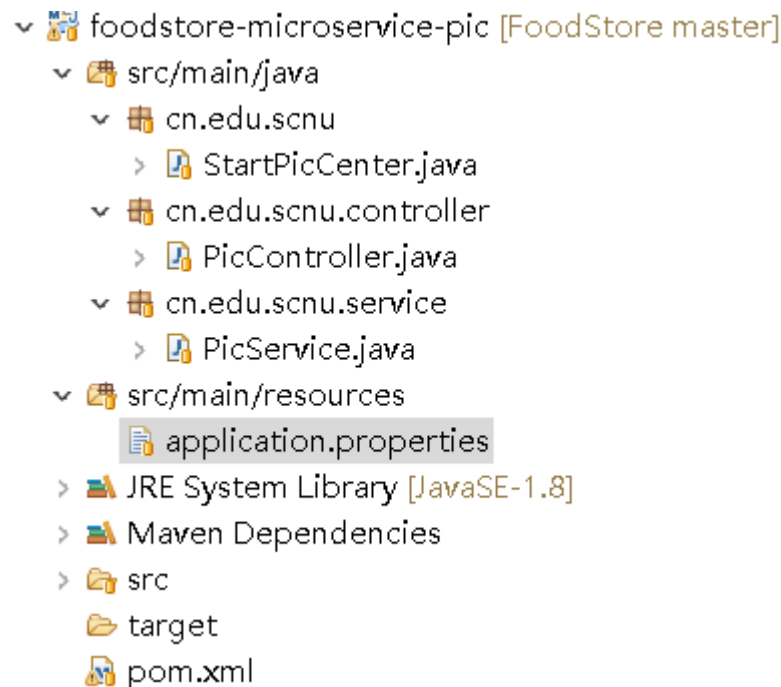
- b) Zuul 项目继承 spring-boot-starter-parent，引入了 spring-cloud-dependencies 管理的依赖，并引入了 eureka 和 zuul 相关启动依赖。其次，其 pom 文件中添加了插件简化资源 spring-boot-maven-plugin。配置文件中，指定了 zuul 的端口号，不同微服务的路由，并将标头设置为空。Zuul 作为项目与外界的接口，承受连接压力大，需要做集群。把项目打成 jar 包后，通过命令指定不同端口来运行，即可完成集群搭建。

#### 4. 商品微服务搭建

- ▼  foodstore-microservice-product [FoodStore master]
  - ▼  src/main/java
    - ▼  cn.edu.scnu
      - >  StarterProductCenter.java
    - ▼  cn.edu.scnu.product.controller
      - >  ProductController.java
    - ▼  cn.edu.scnu.product.mapper
      - >  ProductMapper.java
    - ▼  cn.edu.scnu.product.service
      - >  ProductService.java
  - ▼  src/main/resources
    - ▼  mapper
      -  ProductMapper.xml
      -  backup-application.properties
      -  bootstrap.properties
    - >  JRE System Library [JavaSE-1.8]
    - >  Maven Dependencies
    - >  src
    -  target
    -  pom.xml

- a)
- b) Foodstore-microservice-product 微服务继承了 foodstore-parent 父级工程，将项目管理的三个公共资源全部引入，添加了 redis 的简化依赖以及 spring-cloud-starter-config 启动依赖。
- c) 该微服务编写了启动类 StarterProductCenter.java，分了控制层、业务层、持久层，完成了商品查询、商品分类查询、后台商品新增、后台商品信息更新、后台商品删除功能。

## 5. 商品图片微服务搭建



a)

b) Foodstore-microservice-pic 微服务继承了 foodstore-parent 父级工程并引入了 foodstore-common-resources 公共资源。

c) 该微服务编写了启动类 StartPicCenter.java，分了控制层与业务层，完成了图片的上传功能。配置文件中指定了端口号，微服务名称，图片存储路径以及图片的 url 前缀。

## 6. config 配置中心的搭建：

(1) 设置配置文件，通过访问 git 读取配置，将配置中心注册在 eureka 中心，形成服务提供者，为其他服务提供高可用的访问获取 git 中 properties 的配置文件资源。

```
server.port=11000
spring.cloud.config.server.git.uri=https://gitee.com/bayan01/fsconfig
spring.cloud.config.server.git.searchPaths=/xutest
spring.cloud.config.label=master
spring.cloud.config.server.git.username=bayan01
spring.cloud.config.server.git.password=*****

#eureka config server
spring.application.name=configserver
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka ,http://localho:
```

```
1 spring.datasource.driver-class-name=com.mysql.jdbc.Driver
2 spring.datasource.url=jdbc:mysql://114.116.102.86:8066/easydb?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true
3 spring.datasource.username=root
4 spring.datasource.password=root
5 spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
6 spring.datasource.initialSize=5
7 spring.datasource.maxActive=50
8 spring.datasource.maxIdle=10
9 spring.datasource.minIdle=5
10 mybatis.typeAliasesPackage=com.easymall.common.pojo
11 mybatis.mapperLocations=classpath:mapper/*.xml
12 mybatis.configuration.mapUnderscoreToCamelCase=true
13 mybatis.configuration.cacheEnabled=false
```

```
#redis config
# spring.redis.cluster.nodes=192.168.126.151:8000,192.168.126.151:8001,192.168.126.151:8002
spring.redis.cluster.nodes=114.115.173.107:8000,114.115.173.107:8001,114.115.173.107:8002
spring.redis.cluster.maxTotal=200
spring.redis.cluster.maxIdle=8
spring.redis.cluster.minIdle=2
```

(2) 将其他微服务整合到 config 配置中心作为其客户端。

## 7. 公用工程 rediscluster

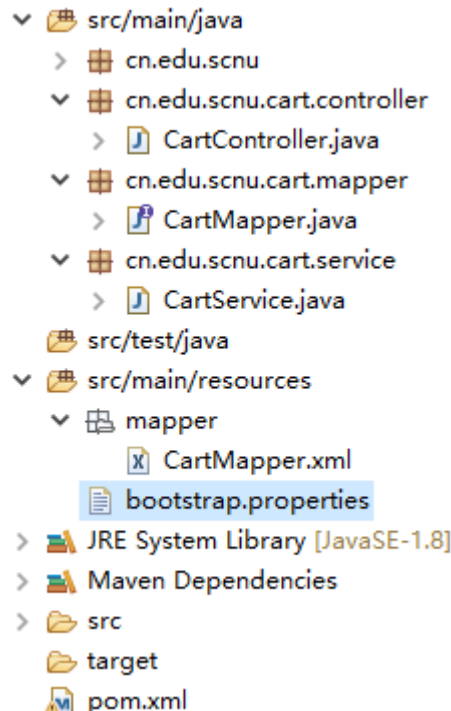
(1) 将操作 rediscluster 集群的工作整合到 springboot, 配置连接信息, 并创建配置类。

```
18 public class RedisClusterConfig {
19     private List<String> nodes;
20     private Integer maxTotal;
21     private Integer maxIdle;
22     private Integer minIdle;
23     //初始化JedisCluster的方法
24     @Bean
25     public JedisCluster initJedisCluster(){
26         //收集节点信息
27         Set<HostAndPort> set=new HashSet<HostAndPort>();
28         for (String node : nodes) {
29             //node="114.115.173.107:8000"
30             String host=node.split(":")[0];
31             Integer port=Integer.
32                 parseInt(node.split(":")[1]);
33             set.add(new HostAndPort(host,port));
34         }
35         GenericObjectPoolConfig config=new GenericObjectPoolConfig();
36         config.setMaxTotal(maxTotal);
37         config.setMaxIdle(maxIdle);
```

(2) Jedis 通过一个节点从 redis 服务器获取整个集群的信息, 这些信息包括节点与其对应连接池的映射关系和槽位与槽位所在节点对应连接池的映射信息, 将信息保存在 JedisClusterInfoCache 中。发送请求时, JedisCluster 对象从初始化得到的集群 map 中获取 key 对应的节点连接, 即一个可用的 Jedis 对象, 然后通过这个对象发送操作命令。

## 8. 购物车系统的实现

(1) 配置各种文件



- (2) 建立 dao 层, service 层, controller 层, 实现购物车商品展示, 购物车商品添加, 购物车商品删除, 购物车商品数量修改以及商品批量选择的功能。

```
@RequestMapping("cart/manage")
public class CartController {
    @Autowired
    private CartService cartService;

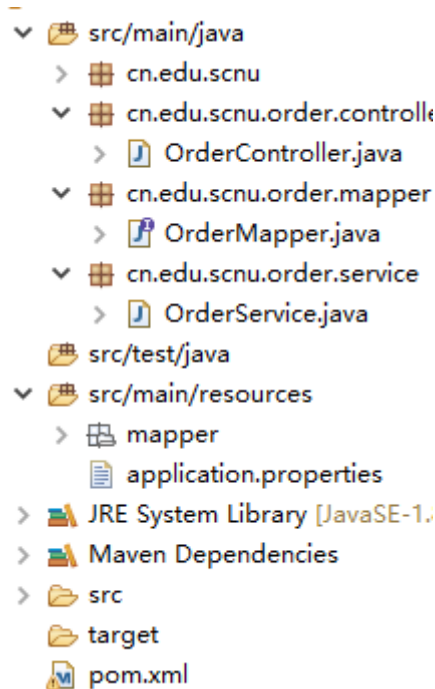
    @RequestMapping("query")
    public List<Cart> queryMyCart(String userId){
        if(!StringUtils.isEmpty(userId)){
            return null;
        }
        return this.cartService.queryMyCart(userId);
    }

    @RequestMapping("save")
    public SysResult cartSave(Cart cart){
        // System.out.println(cart);
        try{
            this.cartService.cartSave(cart);
            return SysResult.ok();
        }catch(Exception e){
            e.printStackTrace();
            return SysResult.build(201, "", null);
        }
    }
}
```

在 controller 层实现查询, 保存, 增加, 删除购物车商品的业务的调度, 之后再 service 层实现查询库存, 核对购物车商品, 更新商品数量, 删除信息等的一系列具体业务逻辑。

## 9. 订单系统的实现

### (1) 配置各种文件



### (2) 建立 dao 层, service 层, controller 层, 实现订单展示, 添加订单, 删除订单的功能。

```
public class OrderController {
    @Autowired
    private OrderService orderService;

    @RequestMapping("save")
    public SysResult saveOrder(Order order){
        //订单增加
        try{
            this.orderService.saveOrder(order);
            return SysResult.ok();
        }catch (Exception e){
            e.printStackTrace();
            return SysResult.build(201, e.getMessage(), null);
        }
    }

    @RequestMapping("query/{userId}")
    public List<Order> queryMyOrder(@PathVariable String userId) {
        return orderService.queryMyOrder(userId);
    }

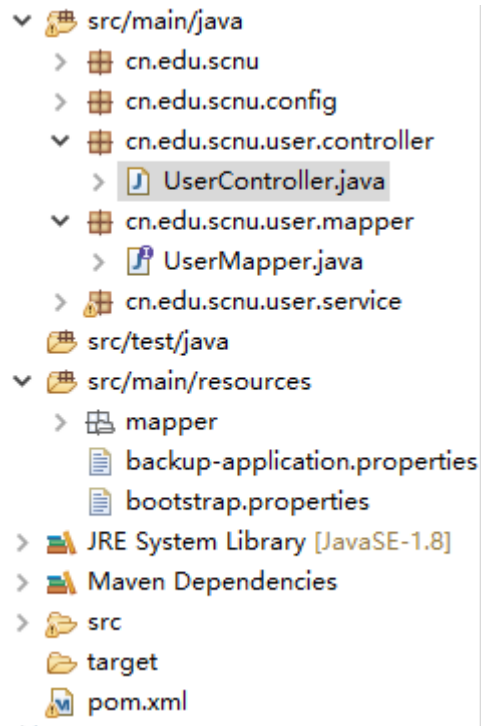
    @RequestMapping("delete/{orderId}")
```

在 controller 层实现订单查询, 保存, 以及订单删除的业务调度, 之后再 service 层实现订单创建, 订单信息处理, 订单删除, 订单落库等的一系列具体业务逻辑。

## 10. 用户系统的实现

### (1) 配置各种文件





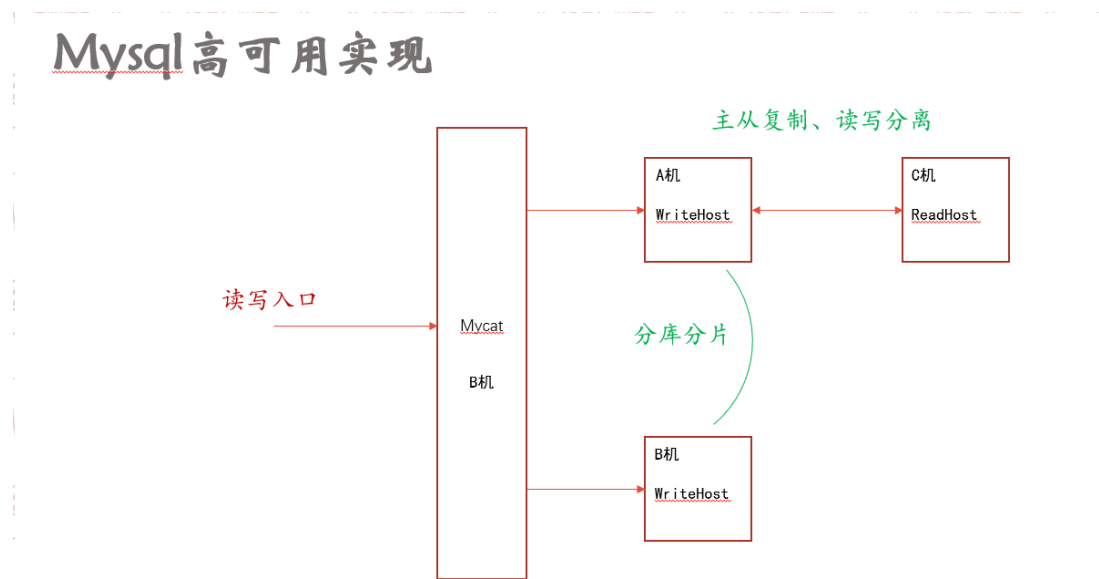
- (2) 建立 dao 层, service 层, controller 层, 实现用户登录超时, 一个账号最多 3 个用户同时登录

```
UserController.java
61     if (StringUtils.isEmpty(ticket)) {
62         CookieUtils.setCookie(request, response, "FD_TICKET", ticket);
63         // System.out.println("ok");
64         return SysResult.ok();
65     } else {
66         return SysResult.build(201, "登录失败", null);
67     }
68 }
69
70 @RequestMapping("/user/manage/query/{ticket}")
71 public SysResult checkLoginUser(@PathVariable String ticket) {
72     String userJson = this.userService.queryUserJson(ticket);
73     if (StringUtils.isEmpty(userJson)) {
74         return SysResult.build(200, "ok", userJson);
75     } else {
76         return SysResult.build(201, "", null);
77     }
78 }
79
80 @RequestMapping("user/manage/logout")
81 public SysResult doLogout(HttpSession httpSession, HttpServletRequest
```

在 controller 层实现账号查询, 账号注册, 账号登录, 账号退出等业务的调度, 在 service 层实现账号登录超时, 一个账号最多三个用户登录, 账号 redis 缓存获取, redis 缓存插入和删除, 账号信息核对, 账号信息入库等的一系列具体业务逻辑。

## 4.2.2Mysql

Mysql 高可用实现总体框架



小组共买了 3 台华为云云上服务器，Mysql 的安装和数据存储都是在这个云上服务器里。

Mycat 中间件作为读写入口可以装在任意一台机，不影响同机的 Mysql 运作，所以这里我安在了 B 机上。Mycat 是数据库中间件，可实现高可用（解决数据量太大，高并发等问题），功能包括：主从自动切换（双机热备）；读写分离（负载均衡）；数据分库分表等。

分库分片：我们将 A 机和 B 机进行分库分片。这里全部表是按字符串类型的 xxid 作为索引分片，采用应用于字符串 murmur 分片算法。当一个数据插入时，根据索引的值再通过算法决定这条数据应插入在哪架机子中，虽然一张表数据分库存储，但最后通过 Mycat 查询表时，是将所有分片过的表合并后返回全部数据。

读写分离：我们将 A 机和 C 机实现读写分离，一个作为写主机、一个作为读主机。实际大型项目中一个写主机配对多个读主机，因为实际应用中查询次数多于增删次数。读写分离分摊了数据库的压力，做到一个机器上数据库专职某项工作，提高整体效率。

下图 Mycat 的 schema.xml 配置高可用

```

<dataNode name="dn1" dataHost="localhost1" database="easydb" />
<dataNode name="dn2" dataHost="localhost2" database="easydb" />

<dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
  writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">
  <heartbeat>select user()</heartbeat>
  <writeHost host="hostM1" url="114.115.149.170:3306" user="root" password="root">
    <readHost host="hostM2" url="114.115.173.107:3306" user="root" password="root" />
  </writeHost>
</dataHost>

<dataHost name="localhost2" maxCon="1000" minCon="10" balance="0"
  writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">
  <heartbeat>select user()</heartbeat>
  <writeHost host="hostM3" url="114.116.102.86:3306" user="root" password="root">
  </writeHost>
</dataHost>

<schema name="easydb" checkSQLSchema="false" sqlMaxLimit="100">
  <table name="t_cart" dataNode="dn1, dn2" rule="user_id_sharding-by-murmur" primaryKey="id" autoIncrement="true" />
  <table name="t_order" dataNode="dn1, dn2" rule="order_id_sharding-by-murmur"/>
  <table name="t_order_item" dataNode="dn1, dn2" rule="sharding-by-murmur"/>
  <table name="t_product" dataNode="dn1, dn2" rule="product_id_sharding-by-murmur"/>
  <table name="t_user" dataNode="dn1, dn2" rule="user_id_sharding-by-murmur"/>
</schema>

```

读写分离

分库分片

分片算法规则

easydb数据库下的5张表

下图 rule.xml 文件里，采用应用于字符串 murmur 分片算法

```

<function name="murmur">
  <class>io.mycat.route.function.PartitionByMurmurHash</class>
  <property name="seed">0</property><!-- 默认是0 -->
  <property name="count">2</property><!-- 要分片的数据库节点数量，必须指定，否则没法分片 -->
  <property name="virtualBucketTimes">160</property><!-- 一个实际的数据库节点被映射为这么多虚拟节点，默认是160倍，也就是虚拟节点数是物理节点数的160倍 -->
  <!-- <property name="weightMapFile">weightMapFile</property> 节点的权重，没有指定权重的节点默认是1，以properties文件的格式填写，以从0开始到count-1的整 -->
  <!-- <property name="bucketMapPath">/etc/mycat/bucketMapPath</property> 用于测试时观察各物理节点与虚拟节点的分布情况。如果指定了这个属性，会把虚拟节点的murmur hash值与物理节点的映射按行输出到这个文件，没有默认值，如果不指定。 -->
</function>

```

读写分离实现效果

8 # C 机

user_id	user_name	user_password	user_nickname
77777777-59e-4aaf-9332-fd7b294bc208	777	202cb962ac59075b964b07152d234b70	gsr
7ad64094-944-4a4c-b2f-41d7dd9fb69	wzm3	202cb962ac59075b964b07152d234b70	test03
eb6a1844-9e6-45d3-b047-1233c7233828	wzm2	202cb962ac59075b964b07152d234b70	test02
f17c01d3-c68-4f34-9134-902081edbde0	wzm1	202cb962ac59075b964b07152d234b70	test01
zzzzzzzz-59e-4aaf-9332-fd7b294bc208	zzz	202cb962ac59075b964b07152d234b70	gsr

9 # B 机

user_id	user_name	user_password	user_nickname
cccccccc-59e-4aaf-9332-fd7b294bc208	ccc	202cb962ac59075b964b07152d234b70	gsr
vvvvvvvv-59e-4aaf-9332-fd7b294bc208	vvv	202cb962ac59075b964b07152d234b70	gsr
(NULL)	(NULL)	" "	上帝

```

1 # MySQL
2
3 SELECT * FROM t_user
4 DELETE FROM t_user WHERE user_nickname = 'gsr'
5
6 INSERT INTO t_user(user_id,user_name,user_password,user_nickname,user_email,user_type)
7 VALUES ('cccccccc-159e-4aaf-9332-fd7b294bc208','ccc',MD5('123'),'gsr','cccccc@163.com',0);
8
9 INSERT INTO t_user(user_id,user_name,user_password,user_nickname,user_email,user_type)
10 VALUES ('77777777-159e-4aaf-9332-fd7b294bc208','777',MD5('123'),'gsr','77777@163.com',0);
11
12 INSERT INTO t_user(user_id,user_name,user_password,user_nickname,user_email,user_type)
13 VALUES ('vvvvvvvv-159e-4aaf-9332-fd7b294bc208','vvv',MD5('123'),'gsr','vvvvv@163.com',0);
14
15 INSERT INTO t_user(user_id,user_name,user_password,user_nickname,user_email,user_type)
16 VALUES ('zzzzzzzz-159e-4aaf-9332-fd7b294bc208','zzz',MD5('123'),'gsr','zzzzz@163.com',0);

```

user_id	user_name	user_password	user_nickname	user_email	user_type
77777777-159e-4aaf-9332-fd7b294bc208	777	202cb962ac59075b964b07152d234b70	gsr	77777@163.com	0
7ad64094-344-4a4c-bd2f-41d7dd9fb69	wzm3	202cb962ac59075b964b07152d234b70	test03	wzm3@qq.com	0
eb6a1844-f9e6-45d3-b047-1233c7233828	wzm2	202cb962ac59075b964b07152d234b70	test02	test02@123.com	0
f17c01d3-1c68-4f34-9134-902081edbd0	wzm1	202cb962ac59075b964b07152d234b70	test01	test01@123.com	0
zzzzzzzz-159e-4aaf-9332-fd7b294bc208	zzz	202cb962ac59075b964b07152d234b70	gsr	zzzzz@163.com	0
cccccccc-159e-4aaf-9332-fd7b294bc208	ccc	202cb962ac59075b964b07152d234b70	gsr	cccccc@163.com	0
vvvvvvvv-159e-4aaf-9332-fd7b294bc208	vvv	202cb962ac59075b964b07152d234b70	gsr	vvvvv@163.com	0

读写分离的基础上要做到主从复制，我们将 A 机和 C 机配置成主从复制效果，无论从两架机的哪架机进行增删改查，都能做到数据同步。

```

mysql>
mysql> show slave status \G;
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: 114.115.173.107
A机跟从C机 Master_User: root
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000005
Read_Master_Log_Pos: 120
Relay_Log_File: mysqld-relay-bin.000012
Relay_Log_Pos: 283
Relay_Master_Log_File: mysql-bin.000005
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:

```

```

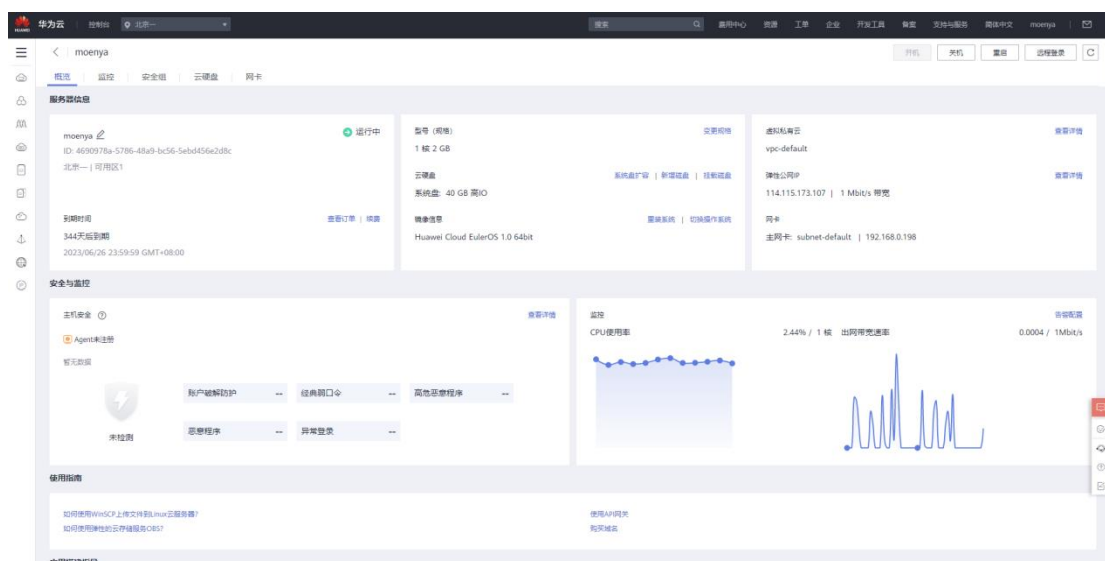
mysql>
mysql> SHOW slave STATUS \G;
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: 114.115.149.170
C机跟从A机 Master_User: root
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000007
Read_Master_Log_Pos: 3123
Relay_Log_File: mysqld-relay-bin.000030
Relay_Log_Pos: 3286
Relay_Master_Log_File: mysql-bin.000007
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:

```

## 4.2.3 Redis

### 1. Redis 相关依赖的下载配置

使用华为云服务器，对云服务器进行配置，安装 Redis，下载相关依赖。服务器预览如下：



配置安全组规则，特别是入方向规则，其中包含允许 xshell 的远程连接 SSH 的端口号以及相应的网卡端口号等协议和端口。

安全组规则						
入方向规则   出方向规则						
安全组名称	优先级	策略	协议端口	类型	源地址	描述
default	1	允许	TCP: 6-36400	IPv4	0.0.0.0/0	--
	1	允许	ICMP: 全部	IPv4	0.0.0.0/0	--
	1	允许	TCP: 3389	IPv4	0.0.0.0/0	Permit default Windows remote desktop p...
	1	允许	TCP: 22	IPv4	0.0.0.0/0	Permit default Linux SSH port.
	100	允许	全部	IPv4	default	--

安装相关依赖略，最终结果如下：

```
redis-3.2.11
ruby-2.5.0
```

## 2. 配置多节点 Redis

如果一个服务器中只启动一个 redis 的 server。不足以充分发挥一个服务器的性能，所以一般在一台服务器上,我们可以根据服务器性能多启动几个 redis 实例,这样一个服务器可以运行多个 Redis 节点，这时要保证多个节点的端口不能冲突。

Redis 运行在保护模式下，保护模式默认是启用的，没有指定绑定地址，没有认证密码要求。在这种模式下，连接只接受环回接口。如果想从外部电脑连接到 Redis 是不行的，远程链接不能进行 CRUD 等操作。为了保证多个节点端口不冲突，要修改 redis 配置文件。

(1) 注释掉 bind，以取消链接限制

```
# bind 127.0.0.1
```

(2) 关闭保护模式，以取消每次连接时都需要提供登陆密码的限制

```
protected-mode no
```

(3) 修改端口

```
# 27 port
port 6379
```

- (4) 设置客户端空闲时间，指定为 3600 秒后才断开

```
timeout 3600
```

- (5) daemonize 设置成 yes，让 redis 服务器启动有守护进程管理

```
daemonize yes
```

- (6) 设置 pid 文件名和 logfile，每一个端口相对应

```
pidfile /var/run/redis_6379.pid
```

```
# Output for logging  
logfile "log6379.log"
```

- (7) 配置 save 策略

```
save 900 1  
save 300 10  
save 60 10000
```

- (8) 指定 dump 的持久化文件，每个服务单独指向一个文件

```
# The filename where to dump the DB  
dbfilename dump6379.rdb
```

- (9) 将该模板保存，并拷贝为 redis01.conf、redis02.conf、redis03.conf 三份，并修改端口号。

```
redis01.conf  
redis02.conf  
redis03.conf
```

- (10) 启动三个节点即可，配置完成。

```
root      1297944      1  0 21:13 ?        00:00:00 redis-server *:6379  
root      1297978      1  0 21:14 ?        00:00:00 redis-server *:6380  
root      1298013      1  0 21:14 ?        00:00:00 redis-server *:6381
```

### 3. 配置 Redis 的一主二从关系

- (1) 配置三个端口的配置文件为 redis6382.conf、redis6383.conf、redis6384.conf

```
redis6382.conf  
redis6383.conf  
redis6384.conf
```

- (2) 启动三个节点

```
[root@hecs-295796 redis-3.2.11]# redis-server redis6382.conf  
[root@hecs-295796 redis-3.2.11]# redis-server redis6383.conf  
[root@hecs-295796 redis-3.2.11]# redis-server redis6384.conf  
[root@hecs-295796 redis-3.2.11]# ps -ef| grep redis  
root      11945      1  0 Jul15 ?        00:00:57 redis-server *:6382  
root      11980      1  0 Jul15 ?        00:00:55 redis-server *:6383  
root      11997      1  0 Jul15 ?        00:00:54 redis-server *:6384
```

- (3) 默认情况下，每个节点都是 master，通过修改配置文件定义主从，并通过命令挂接主节点，显示从节点信息如下：

```
[root@hecs-295796 ~]# redis-cli -p 6382
127.0.0.1:6382> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=114.115.173.107,port=6383,state=online,offset=239352,lag=1
slave1:ip=114.115.173.107,port=6384,state=online,offset=239352,lag=1
master_repl_offset:239352
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:2
repl_backlog_histlen:239351
```

可以看出已经有两个从节点了。

#### 4. 配置 Redis 哨兵监听主从

哨兵监听的主从关系是状态。每一次角色的变化都由哨兵来监控——对所有的主从结构的决策都需要投票。

只有一个哨兵节点的管理结构是不可信，因为如果当哨兵连接主从结构的网络异常时，会误判为主从结构的节点出问题。所以引入哨兵集群。

搭建 3 个哨兵节点，管理 6382,6383/6384 的主从结构。

- (1) 配置三个哨兵的端口 26379 26380 26381

配置 bind 的 ip 地址绑定，设置 protected-mode 为 no，设置端口号，并设置监听主节点的配置核心内容。

```
sentinel01.conf
sentinel02.conf
sentinel03.conf
sentinel.conf
```

- (2) 先启动主从结构中的三个节点：6382,6382.6384，并启动三个哨兵集群，为了查看日志信息，每个哨兵使用一个 Xshell 窗口。可以看出每个哨兵相互发现

```
[root@hecs-295796 redis-3.2.11]# redis-sentinel sentinel02.conf

Redis 3.2.11 (00000000/0) 64 bit

Running in sentinel mode
Port: 26380
PID: 1304025

http://redis.io

1304025:X 17 Jul 21:26:23.553 # Sentinel ID is 735b9a266bc7126d64bd4b0401b8165bcd535e9
1304025:X 17 Jul 21:26:23.553 # +monitor master mymaster 192.168.126.161 6382 quorum 2
1304025:X 17 Jul 21:26:53.586 # +sdown master mymaster 192.168.126.161 6382
1304025:X 17 Jul 21:26:53.586 # +sdown slave 192.168.126.161:6384 192.168.126.161 6384 @ mymaster 192.168.126.161 6382
1304025:X 17 Jul 21:26:53.586 # +sdown slave 192.168.126.161:6383 192.168.126.161 6383 @ mymaster 192.168.126.161 6382
1304025:X 17 Jul 21:26:53.586 # +sdown sentinel 3b9eace86667cd17c58c5fc6f2096466d2cdd534 192.168.126.161 26381 @ mymaster 1
92.168.126.161 6382
1304025:X 17 Jul 21:26:53.586 # +sdown sentinel 676fdda65f18837b227b9209d702994eba23473f 192.168.126.161 26379 @ mymaster 1
92.168.126.161 6382
```

#### 5. 配置 Redis 集群



Redis3.0 有一个新的 redis 集群结构,引入了很多新的概念和逻辑。我们配置该集群以将主从复制、哨兵监听的逻辑整合到了这个该结构中。

- (1) 配置 Ruby, 安装 Redis 接口, 详略。
- (2) 配置集群配置文件, 以 8000-8005 命名。追加模式 AOP 的持久化策略, 开启 AOP 模式的持久化数据, 并指定一个端口号对应的持久化文件; 将节点启动定义为集群模式, 并打开指定一个集群节点运行状态记录文件 conf。

```
8000
8001
8002
8003
8004
8005
```

- (3) 使用 ruby 脚本命令创建一个 3 主且各自有一从的集群结构。最终配置结果如下:

```
[root@hecs-295796 ~]# redis-cli -p 8000
127.0.0.1:8000> cluster info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:6
cluster_size:3
cluster_current_epoch:7
cluster_my_epoch:7
cluster_stats_messages_sent:717603
cluster_stats_messages_received:717603
127.0.0.1:8000> cluster nodes
8387fff5ec5fd56cdec13189b466283a824b3b66 114.115.173.107:8004 slave ed9acc690cde677f662ac6df7fa86a0fd1c4a6d5 0 165823421336
3 2 connected
ed9acc690cde677f662ac6df7fa86a0fd1c4a6d5 114.115.173.107:8001 master - 0 1658234210858 2 connected 5461-10922
6b4f38902780fc4bb8c1726e3b6390ab3001e3d9 192.168.0.198:8000 myself,master - 0 0 7 connected 0-5460
263cf52bf0eaf9ecacf47599ff1ff23cf10ac0bd 114.115.173.107:8005 slave 54957676899ac8ee95fc38036da341cfc7a2bb96 0 165823421236
0 6 connected
967377cc842745ec477a3f11f81ad9645c9789bf 114.115.173.107:8003 slave 6b4f38902780fc4bb8c1726e3b6390ab3001e3d9 0 165823421536
7 7 connected
54957676899ac8ee95fc38036da341cfc7a2bb96 114.115.173.107:8002 master - 0 1658234214363 3 connected 10923-16383
127.0.0.1:8000>
```

## 6. Redis 整合登录

通过点击登录发起 ajax 请求,将用户名传递给后台处理, 根据响应结果判断处理, 如果成功跳转向首页。

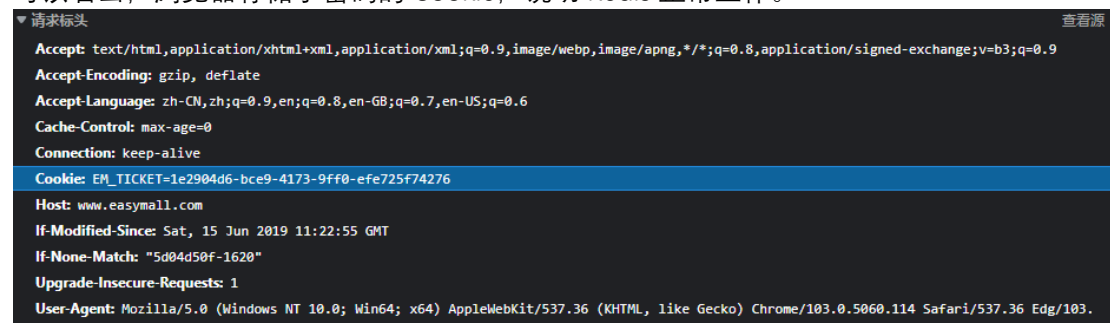
- (1) 在用户中心的 pom 文件添加 redis 依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-redis</artifactId>
  <version>1.4.7.RELEASE</version>
</dependency>
```

- (2) 修改 UserController、UserService、UserMapper 和映射文件 UserMapper。
- (3) 在 Zuul 网关的 application.properties 中添加敏感头为空的配置:  
zuul.sensitive-headers=
- (4) 配置 properties 属性值, 编写配置类, 编写一个连接池的初始化方法。
- (5) 修改 User 的登录逻辑。



可以看出，浏览器存储了密码的 Cookie，说明 Redis 正常工作。



```
▼ 请求标头 查看源
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
Cache-Control: max-age=0
Connection: keep-alive
Cookie: EM_TICKET=1e2904d6-bce9-4173-9ff0-efe725f74276
Host: www.easymall.com
If-Modified-Since: Sat, 15 Jun 2019 11:22:55 GMT
If-None-Match: "5d04d50f-1620"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.114 Safari/537.36 Edg/103.0.1024.62
```

## 5.项目分工说明

### 5.1 组员分工

小组共有五名成员，组长为顾思睿，成员有黄海峰、李牧林、王哲孟、何昕宇。

顾思睿：作为组长规划整个项目进程和审核。负责在华为云服务器安装部署 Mysql、Mycat，实现主从复制、读写分离、分库分片功能。

黄海峰：负责 SpringCloud 框架中依赖资源的管理，高可用双注册中心的搭建，高可用 zuul 网关集群的实现，商品微服务、商品图片微服务的搭建，以及 Nginx 的搭建与配置。

李牧林：FoodStore 生鲜商城前端代码的编写、调试和修改，以及和负责后端的同学沟通如何进行前后端的交互。

王哲孟：负责 config 配置中心的搭建，公用工程 rediscluster，购物车系统，订单系统以及用户微服务的实现。

何昕宇：负责华为云服务器上的 Redis 主从复制、多节点、哨兵的配置，配置 Redis 集群进行管理，配置完成后整合登录功能使得密码存储在浏览器 cookie 中。